CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE143: COMPUTER CONCEPTS & PROGRAMMING

July – November 2019

UNIT – 2

Constants, Variables & Data Types in 'C'

DEPSTAR

Devang Patel Institute of Advance Technology and Research

Objectives

- To be able to create and use variables and constants.
- To be able to distinguish the constants of different data type.
- To be able to name, declare, initialize and assign values to variables.
- To become familiar with fundamental data types.
- To be able to list, describe, and use the C basic data types.

Introduction

- In this chapter, we will discuss
  - Character set
  - C Tokens
  - Constants (integer, real, character, string, enum), symbolic constants
  - Variables (name, declare, assign)
  - Fundamental data type (int, float, double, char, void)

Character Set

- Every language contains a set of characters used to construct words, statements etc.
- C language character set contains the following set of characters...
  - Alphabets
  - Digits
  - Special Symbols
  - White spaces
    - Compiler ignores white spaces unless they are part of a string constant.
    - White spaces may be used to separate words but prohibited between characters of keywords and identifiers.

Character Set

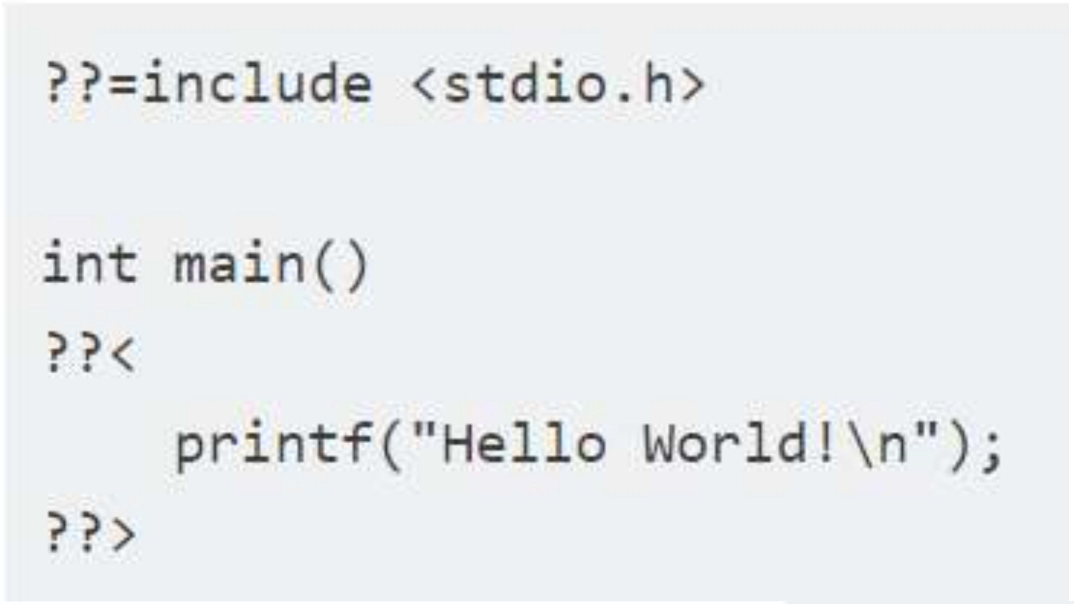
- LETTERS: Uppercase A....Z, lower case a..z
- DIGITS: All decimal digits 0..9
- SPECIAL CHARACTERS: comma(,), period(.), semicolon(;), colon(:), question mark(?), quotation("), dollar sign($), slash(/),back slash(\), percent sign(%), underscore(_), ampersand(&), asterisk(*), number sign(#).
- WHITE SPACES: Blank space, Horizontal tab, Carriage return, Newline, Form feed.

Trigraph characters

- C introduces the concept of trigraph sequences to provide a way to enter certain characters that are not available on some keywords.
- Each Trigraph sequence consists of three characters, 2 question marks followed by another character.

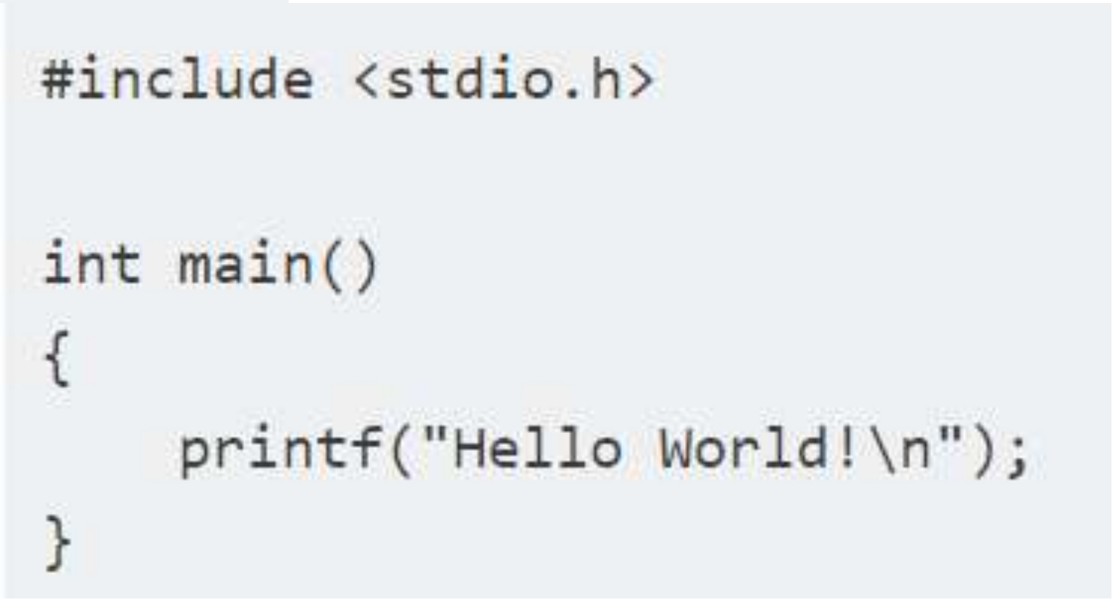
Trigraph sequence	Equal character
??=	#
??(	[
??)	]
??/	\
??<	{
??>	}
??!	|
??’	^
??-	~

Trigraph characters



```
??=include <stdio.h>

int main()
??<
    printf("Hello World!\n");
??>
```



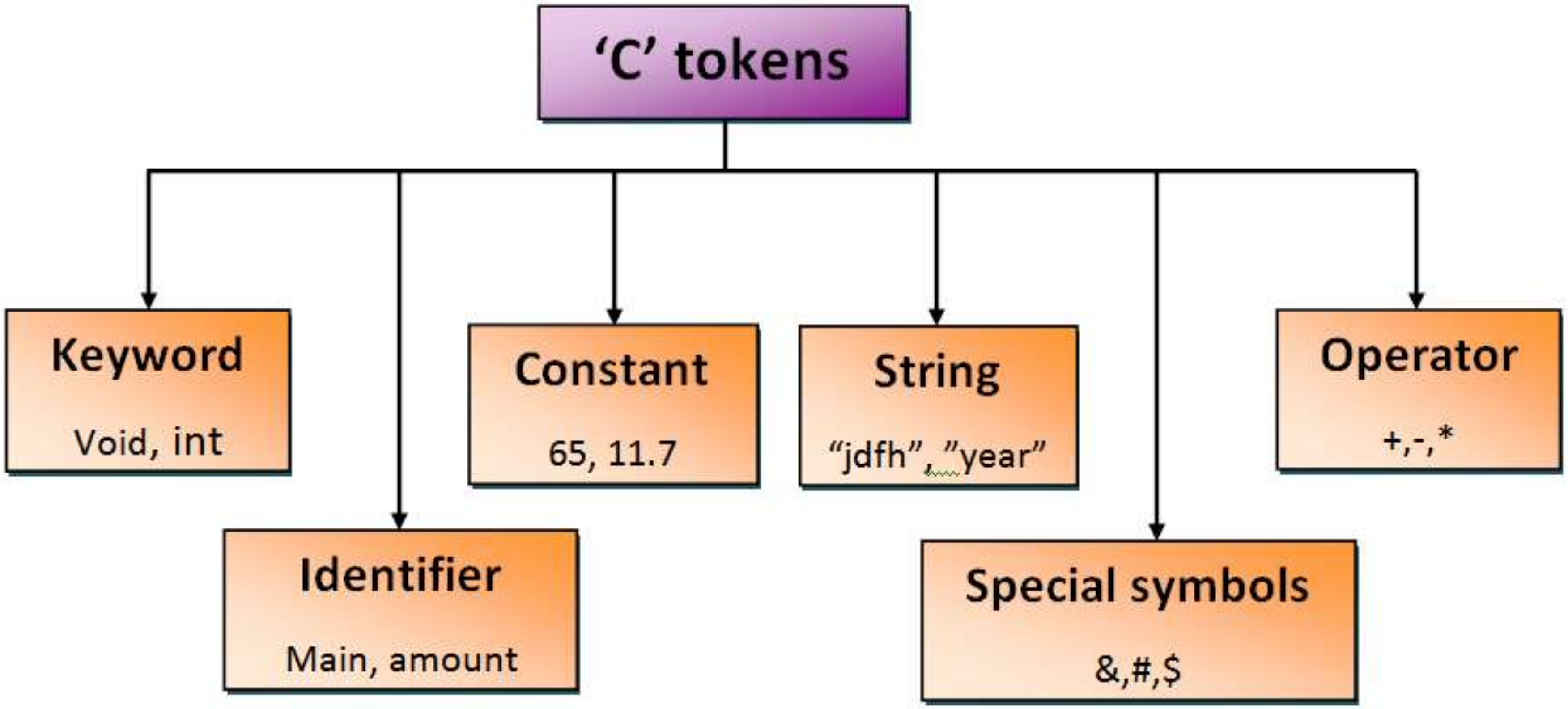
```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

C Tokens

- Every C program is a collection of instructions and every instruction is a collection of some individual units.
- Every smallest individual unit of a c program is called token.
- Every instruction in a c program is a collection of tokens.
- Tokens are used to construct c programs and they are said to the basic building blocks of a c program.

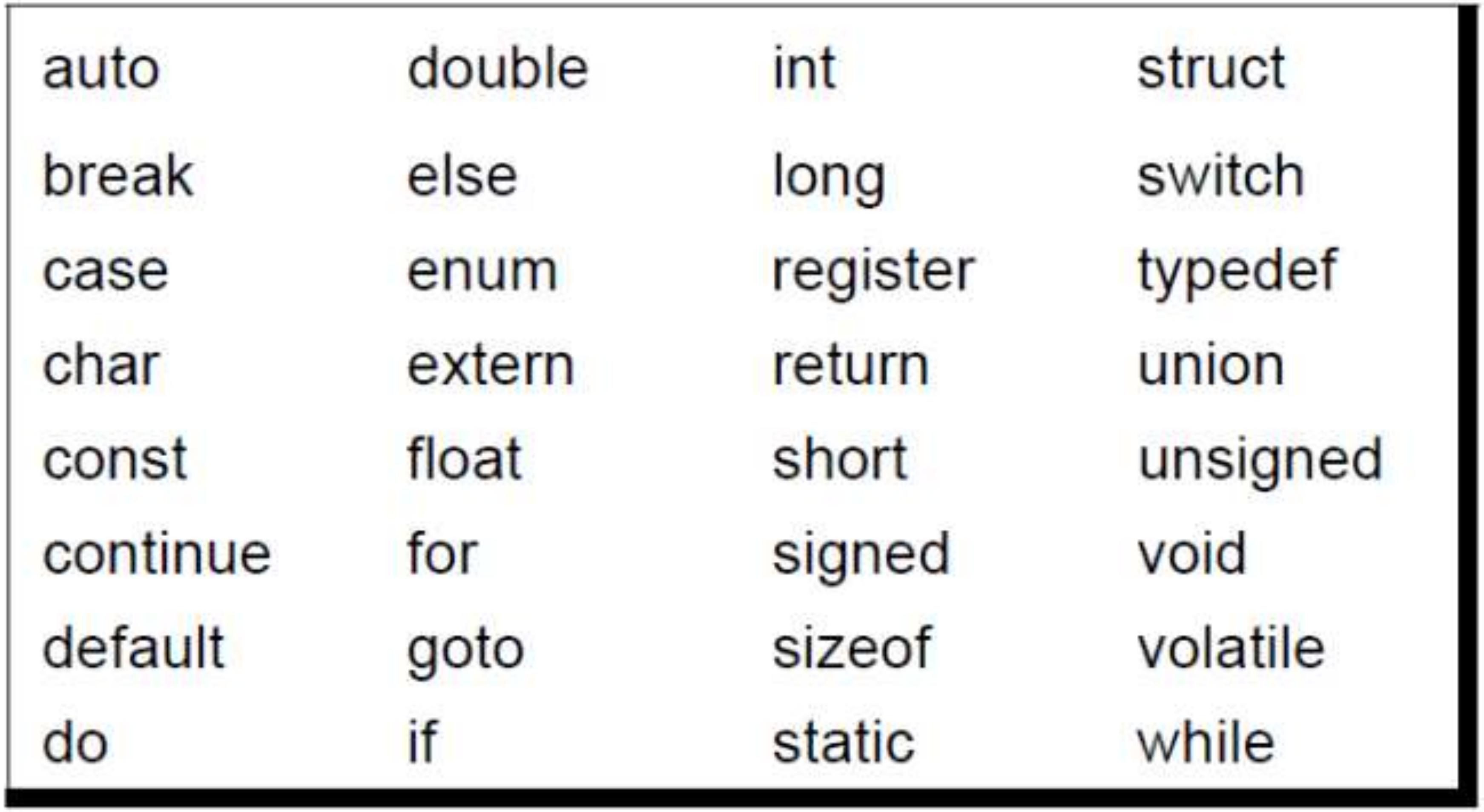
C Tokens


'C' tokens
Keyword
Void, int
Identifier
Main, amount
Constant
65, 11.7
String
"jdfh", "year"
Special symbols
&,#,$
Operator
+,-,*

Keywords

- Keywords are the reserved words with predefined meaning which already known to the compiler.
- Properties of Keywords
    - All the keywords are defined as lowercase letters so they must be use only in lowercase letters
    - Every keyword has a specific meaning, users can not change that meaning.
    - Keywords can not be used as user defined names like variable, functions, arrays, pointers etc...
    - Every keyword represents something or specifies some kind of action to be performed by the compiler.

Keywords



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers

- Identifier is a user defined name of an entity to identify it uniquely during the program execution.
- Example:

```
int marks;
char studentName[30];
```

- Here, **marks** and **studentName** are identifiers.

Identifiers

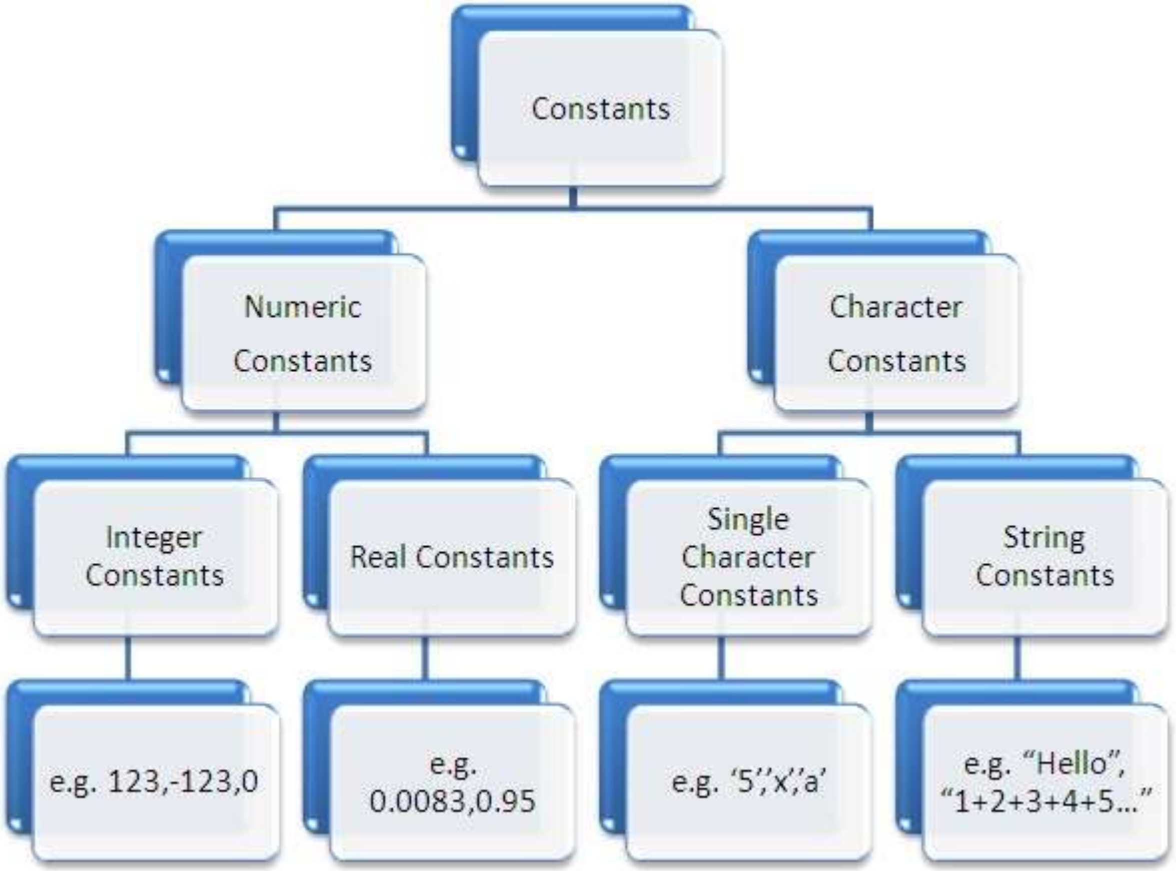
- Rules for creating Identifiers
  - An identifier can contain letters (UPPERCASE and lowercase), numerics & underscore symbol only.
  - An identifier should not start with numerical value. It can start with a letter or an underscore.
  - We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
  - Keywords should not be used as identifiers.
  - There is no limit for length of an identifier. However, compiler consider first 31 characters only.
  - An identifier must be unique in its scope.

Examples of Valid and Invalid Names

Valid Names		Invalid Name	
a	// Valid but poor style	$sum	// $ is illegal
student_name		2names	// First char digit
_aSystemName		sum-salary	// Contains hyphen
_Bool	// Boolean System id	stdnt Nmbr	// Contains spaces
INT_MIN	// System Defined Value	int	// Keyword

Constants

- A constant is a named memory location which holds only one value throughout the program execution.



Creating constants in C

- In c programming language, constants can be created using two concepts...
    - Using 'const' keyword
    - Using '#define' preprocessor

Creating constants in C

- Using 'const' keyword
  - To create a constant, we prefix the variable declaration with 'const' keyword.
  - The general syntax for creating constant using 'const' keyword is as follows...

```
const datatype constantName ;
                    OR
const datatype constantName = value ;
```

  - Example

```
const int x = 10 ;
```

    - Here, 'x' is a integer constant with fixed value 10.

Creating constants in C

- Using '#define' preprocessor
  - To create constant using this preprocessor directive it must be defined at the beginning of the program (because all the preprocessor directives must be written before the gloabal declaration).
  - We use the following syntax to create constant using '#define' preprocessor directive…

```
#define CONSTANTNAME value
```

  - Example:

```
#define PI 3.14
```

    - Here, PI is a constant with value 3.14

Constants in C- Example Program

//Using 'const' keyword

```
#include <stdio.h>
void main()
{
  int i = 9 ;
  const int x = 10 ;

  i = 15 ;
  x = 100 ; // creates an error
  printf("i = %d\n x = %d", i, x ) ;
}
```

- The above program gives an error because we are trying to change the constant variable value (x = 100).

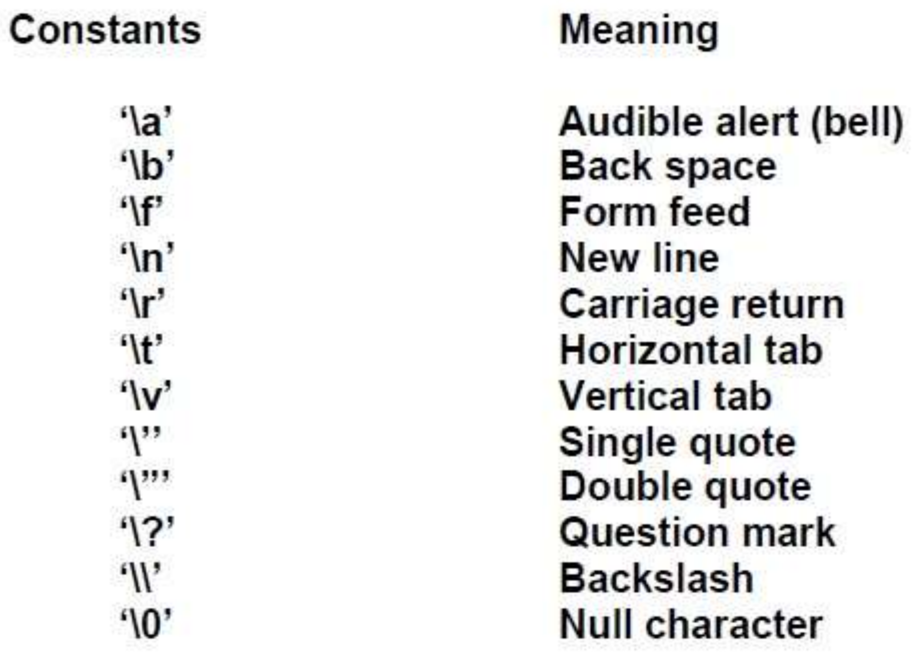
//Using '#define' preprocessor

```
#include <stdio.h>
#define  PI  3.14
void main()
{
  int r, area ;
  printf("Please enter the radius of circle : ") ;
  scanf("%d", &r) ;

  area = PI * (r * r) ;
  printf("Area of the circle = %d", area) ;
}
```

Backslash character constants

- C supports special backslash character constants that are used in output functions.
- These character combinations are known as escape sequences.



Constants	Meaning
‘\a’	Audible alert (bell)
‘\b’	Back space
‘\f’	Form feed
‘\n’	New line
‘\r’	Carriage return
‘\t’	Horizontal tab
‘\v’	Vertical tab
‘\’’	Single quote
‘\”’	Double quote
‘\?’	Question mark
‘\\’	Backslash
‘\0’	Null character

Variables

- Variable is a name given to a memory location where we can store different values of same datatype during the program execution.
- It must be declared in the declaration section before it is used.
- It must have a datatype that determines the range and type of values to be stored.
- A variable name may contain letters, digits and underscore symbol.

Variables

- The following are the rules to specify a variable name...
  - Variable name should not start with digit.
  - Keywords should not be used as variable names.
  - Variable name should not contain any special symbols except underscore(_).
  - Variable name can be of any length but compiler considers only the first 31 characters of the variable name.

- Valid variables are:
  - john
  - x1
  - T_raise
  - first_tag

- Invalid variables are:
  - 123
  - (area)
  - 25th
  - price$
  - %

Declaration of Variables

- Declaration allocate required amount of memory with specified variable name and datatype values into that memory location.
- The declaration can be performed either before the function as global variables or inside any block or function.
- It must be at the beginning of block or function.
- Syntax:

```
Datatype   variableName;
```

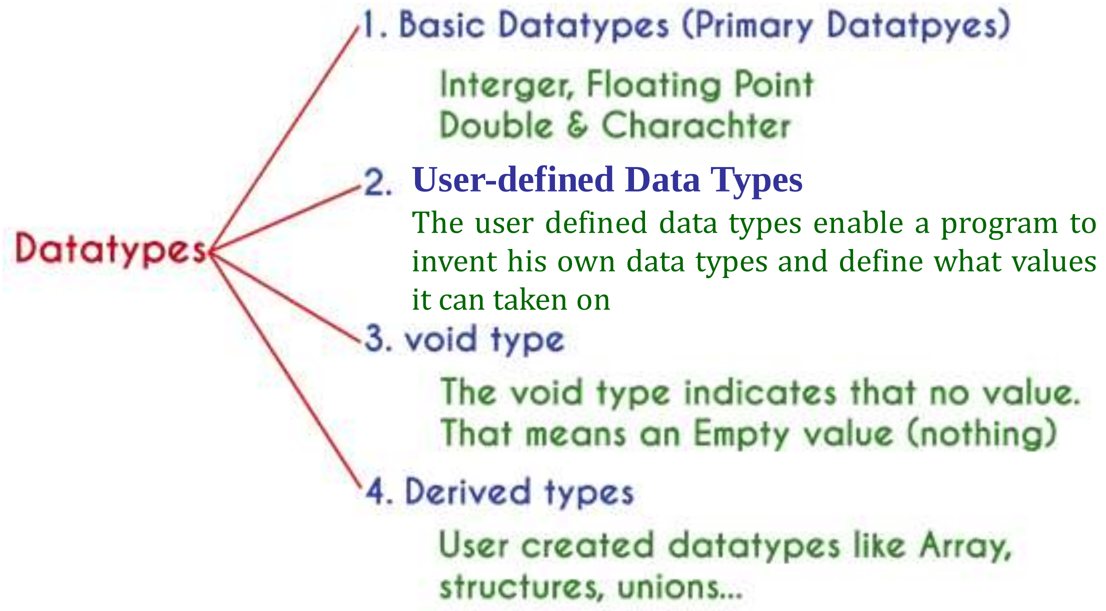
- Example

```
int number;
```

  - The above declaration allocate 2 bytes(32 bit Compiler)/4 bytes(64 bit Compiler) of memory with the name number and allows only integer values into that memory location.

Datatypes

- Datatype is a set of value with predefined characteristics.
- Datatypes are used to declare variable, constants, arrays, pointers and functions.
- The memory size and type of value of a variable are determined by varible datatype.



Datatypes

1. Basic Datatypes (Primary Datatpyes)

   Interger, Floating Point
   Double & Charachter

2. **User-defined Data Types**

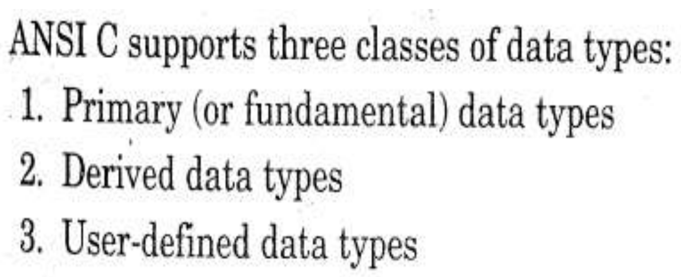
   The user defined data types enable a program to invent his own data types and define what values it can taken on

3. void type

   The void type indicates that no value.
   That means an Empty value (nothing)

4. Derived types

   User created datatypes like Array, structures, unions...

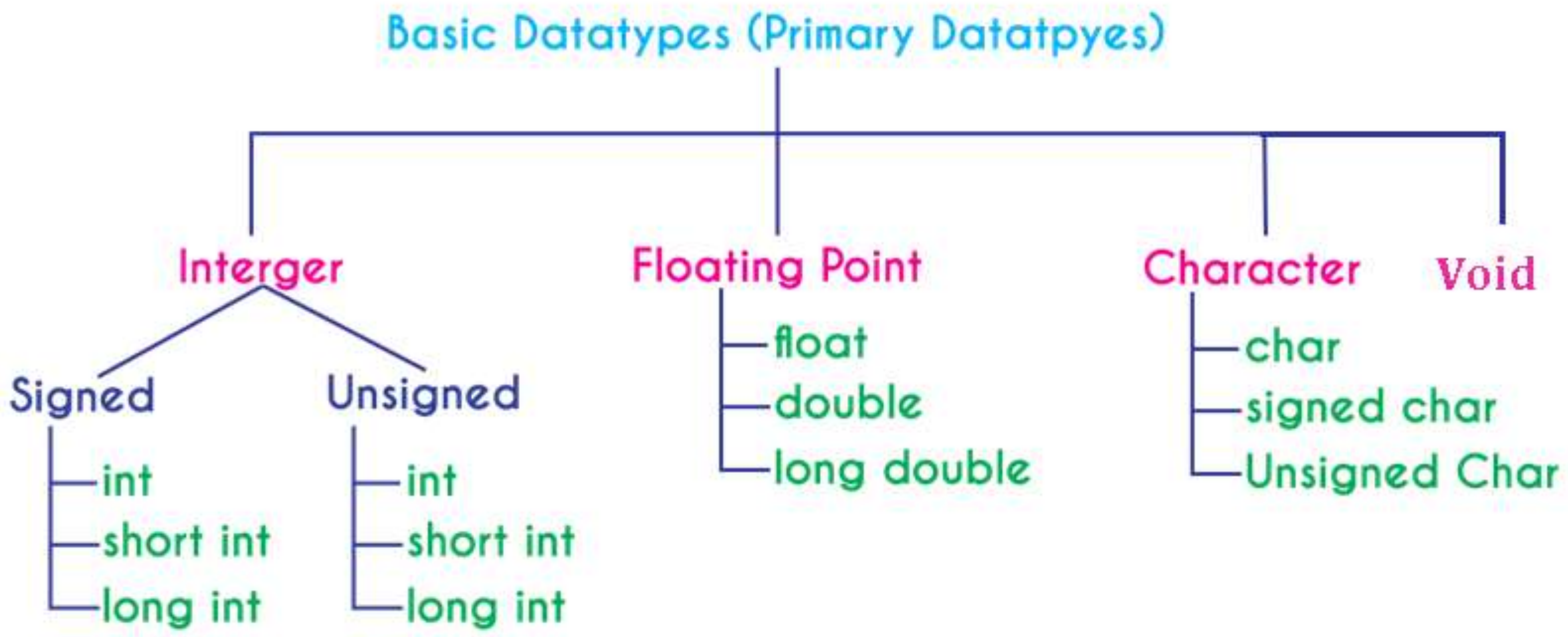


ANSI C supports three classes of data types:

1. Primary (or fundamental) data types
2. Derived data types
3. User-defined data types

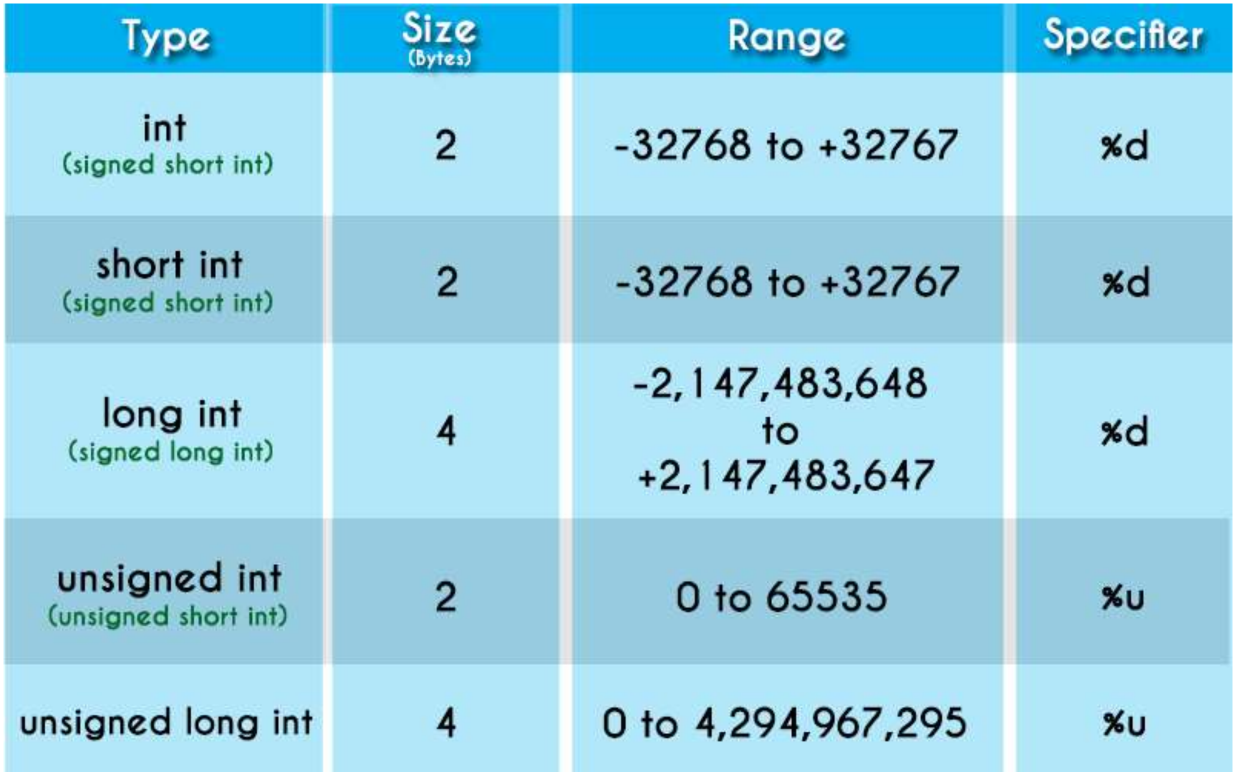
Datatypes- Primary Datatypes

- All the primary datatypes are already defined in the system.
- Primary datatypes are also called as Built-In datatypes.
- The following are the primary datatypes in c programming lanuage...



Datatypes- Integer Datatype

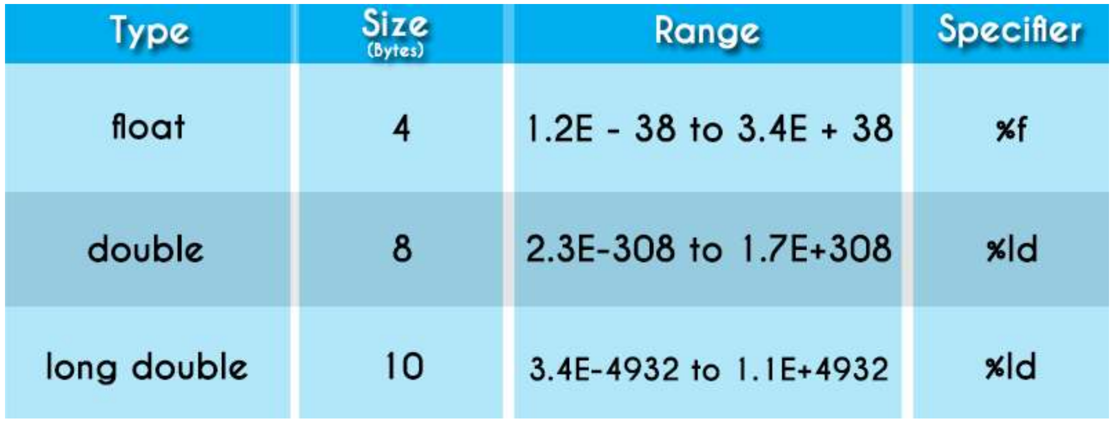
- The keyword "int" is used to represent integer datatype in c.
- The integer datatype is used with different type modifiers like short, long, signed and unsigned.
- The following table provides complete details about integer datatype.



Type	Size (Bytes)	Range	Specifier
int (signed short int)	2	-32768 to +32767	%d
short int (signed short int)	2	-32768 to +32767	%d
long int (signed long int)	4	-2,147,483,648 to +2,147,483,647	%d
unsigned int (unsigned short int)	2	0 to 65535	%u
unsigned long int	4	0 to 4,294,967,295	%u

Datatypes- Floating Point Datatypes

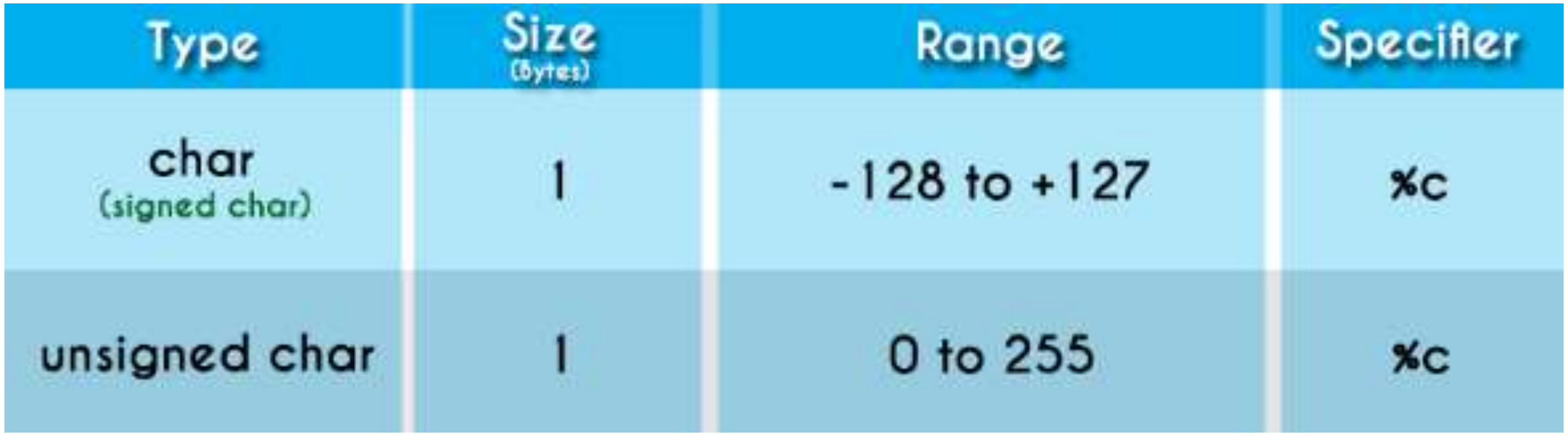
- Floating point datatypes are set of numbers with decimal value.
- The floating point datatype has two variants...
  - float
  - double
- Both float and double are similar but differ in number of decimal places.
  - float value contains 6 decimal places.
  - double value contains 15 or 19 decimal places.
- The following table provides complete details about floating point datatypes.



Type	Size (Bytes)	Range	Specifier
float	4	1.2E - 38 to 3.4E + 38	%f
double	8	2.3E-308 to 1.7E+308	%ld
long double	10	3.4E-4932 to 1.1E+4932	%ld

Datatypes- Character Datatype

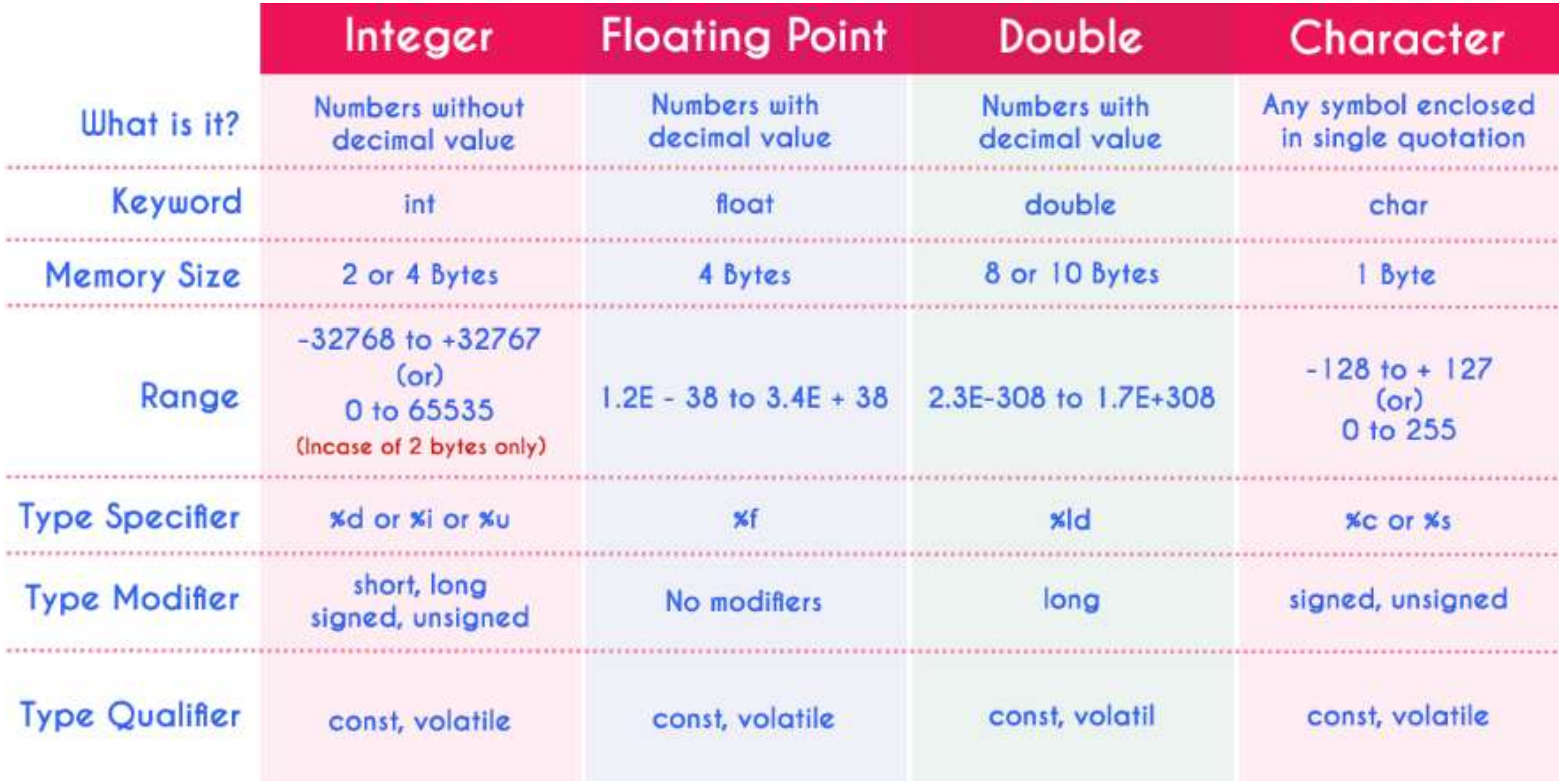
- Character datatype is a set of characters enclosed in single quotations.
- The following table provides complete details about character datatype.



Type	Size (Bytes)	Range	Specifier
char (signed char)	1	-128 to +127	%c
unsigned char	1	0 to 255	%c

Datatypes

- The following table provides complete information about all the datatypes in c programming language.



	Integer	Floating Point	Double	Character
What is it?	Numbers without decimal value	Numbers with decimal value	Numbers with decimal value	Any symbol enclosed in single quotation
Keyword	int	float	double	char
Memory Size	2 or 4 Bytes	4 Bytes	8 or 10 Bytes	1 Byte
Range	-32768 to +32767 (or) 0 to 65535 (Incase of 2 bytes only)	1.2E - 38 to 3.4E + 38	2.3E-308 to 1.7E+308	-128 to + 127 (or) 0 to 255
Type Specifier	%d or %i or %u	%f	%ld	%c or %s
Type Modifier	short, long signed, unsigned	No modifiers	long	signed, unsigned
Type Qualifier	const, volatile	const, volatile	const, volatil	const, volatile

Datatypes- void Datatype

- The void datatype means nothing or no value.
- Generally, void is used to specify a function which does not return any value.
- We also use the void datatype to specify empty parameters of a function.

Declaration of Storage Class

- Storage class in C decides the part of storage to allocate memory for a variable, it also determines the scope of a variable.
- All variables defined in a C program get some physical location in memory where variable's value is stored.
- Memory and CPU registers are types of memory locations where a variable's value can be stored.
- The storage class of a variable in C determines the life time of the variable if this is 'global' or 'local'.
- Along with the life time of a variable, storage class also determines variable's storage location (memory or registers), the scope (visibility level) of the variable, and the initial value of the variable.

Declaration of Storage Class

Storage Class	Declaration Location	Scope (Visibility)	Lifetime (Alive)
auto	Inside a function/block	Within the function/block	Until the function/block completes
register	Inside a function/block	Within the function/block	Until the function/block completes
extern	Outside all functions	Entire file plus other files where the variable is declared as extern	Until the program terminates
static (local)	Inside a function/block	Within the function/block	Until the program terminates
static (global)	Outside all functions	Entire file in which it is declared	Until the program terminates

Declaration of Storage Class

- The general form of a variable declaration that uses a storage class is shown here:

  storage_class_specifier *data_type variable_name;*

- At most one storage class specifier may be given in a declaration.
- If no storage class specifier is specified then following rules are used:
  - Variables declared inside a function are taken to be **auto.**
  - Functions declared within a function are taken to be **extern.**
  - Variables and functions declared outside a function are taken to be **static**, with external linkage.

Assigning values to variables

- The general form of a variable declaration that uses a storage class is shown here:
- The syntax is

  Variable_name=constant

- Example:

  int a=20;

  bal=75.84;

  yes='x';

- C permits multiple assignments in one line.
- Example:

  initial_value=0;final_value=100;

Reading data from keyword

- Another way of giving values to variables is to input data through keyboard using the scanf function.
- The general format of scanf is as follows.

```
scanf(“control string”,&variable1,&variable2,….);
```

- The ampersand symbol & before each variable name is an operator that specifies the variable name’s address.
- Example:

```
scanf(“%d”,&number);
```

#define Directive

- The #define directive allows the definition of macros within your source code.
- These macro definitions allow constant values to be declared for use throughout your code.
- The syntax for creating a constant using #define in the C language is:

#define CNAME value

OR

#define CNAME (expression)

value:
The value of the constant.

CNAME:
The name of the constant. Most C programmers define their constant names in uppercase, but it is not a requirement of the C Language.

expression:
Expression whose value is assigned to the constant.
The expression must be enclosed in parentheses if it contains operators.

#define Directive - Example

```
#include <stdio.h>

#define NAME "Charusat"
#define YEAR 2000

int main()
{
  printf("%s is established in year %d\n", NAME, YEAR);
  return 0;
}
```

Difference between #define & const

- The big advantage of const over #define is type checking.
- We can typecast them and any other thing that can be done with a normal variable.
- One disadvantage that one could think of is extra space for variable which is immaterial due to optimizations done by compilers.
- In general const is a better option if we have a choice. There are situations when #define cannot be replaced by const. For example, #define can take parameters. #define can also be used to replace some text in a program with another text.

Declaring a variable as volatile

- ANSI standard defines another qualifier, **volatile** that could be used to tell explicitly the compiler that a variable's value may be changed at any time by some external sources.
- By declaring a variable as **volatile**, its value may be changed at any time by some external source.
- Example:

```
volatile int date;
```

- The date may be altered by some external factors even if it does not appear on the left hand side of assignment statement.

Overflow and Underflow of Data

- Based on the storage size of data types in any particular system there is a fixed range of values that can be stored in a variable of a given data types.
- **Overflow:** The assignment of a value larger than the storage capacity of the data type of a variable can hold.
- **Underflow:** The assignment of a value smaller than the storage capacity of the data type of a variable can hold.
- C does not provide any warning or indication of Overflow and underflow, It simply gives incorrect results.
- So, It is recommended to define correct data type for handling the input/output values.

Datatypes- Derived Datatypes

- Derived datatypes are user-defined data types.
- The derived datatypes are also called as user defined datatypes or secondary datatypes.
- In c programming language, the derived datatypes are created using the following concepts...
  - Arrays
  - Structures
  - Unions
  - Enumeration

Previous Year Questions

- Classify following variables into valid & invalid variable name in C.

  %, (area), mark, 25th, 123, distance
- What are the advantages of user defined function?
- Classify between valid and Invalid variable names?

  RAM, ANSI, GJ@India, Float variable, One_int, ID#22
- Give difference between variables and constants.

Previous Year Questions

- Why and When do we use the #define directive? OR Explain #define?
- What are trigraph characters? How are they useful?
- Explain C Tokens with example?
- Explain three types of datatypes with example?
- Is main() a user defined function? if yes how does it differ from other user-defined function? if No, give reason for it.